关联规则分析与 Apriori 算法

CS245　数据科学基础　陆朝俊
叶泽林 515030910468

1　问题描述

关联规则分析是数据挖掘中活跃的研究方法之一，其目的是在一个数据集中找出各项之间的关联关系（这种关系一般没有在数据中直接表示出来）。Apriori 是关联规则分析中最常用也是最经典的挖掘频繁项集的算法，在本次作业中，我将实现 Apriori 算法，从交易数据集中发现频繁项集，并生成相应的关联规则。

2　解决方案[1]

2.1　数据集的准备

为验证 Apriori 算法实现的正确性，我根据一定的规则生成了一个交易数据集，数据集的每个实例代表一条交易记录，共 100 个实例。我将商品分为食品（bread、milk、apple、orange、beer）电器（TV、PC、phone、fridge、ele_oven）和工具（scissors、stapler、plate、knife、glue）三类，各按照一定的出现及组合概率生成相应的商品交易记录，具体的参数设定可参见附录 A.1。

我认为对于交易情况的分析，仅使用模拟生成的数据集难以取得真实的结果，因此我额外在 Groceries 数据集上执行了 Apriori 算法。Groceries 数据集是内置于 R 语言的关联分析数据集，来源于某杂货店一个月的真实交易记录，包含 9835 条交易记录及 169 种商品。我将其从 R 语言包中提取出并重组为.csv 格式，再使用 Apriori 算法进行分析。

2.2　Apriori 算法

Apriori 算法是最经典的挖掘频繁项集的算法，实现了在大数据集上可行的关联规则提取，其核心思想是**通过连接产生候选项与其支持度，然后通过剪枝生成频繁项集**，步骤主要为：

1. 找出所有的频繁项集（支持度大于等于给定的阈值）；
2. 由频繁项集产生强关联规则（经过上个步骤后满足给定的置信度阈值的规则）。

为验证 Apriori 算法实现的正确性，我先使用蛮力算法在模拟交易数据集上运行一次，将结果与 Apriori 算法的结果进行比较。验证算法的正确性后，在 Groceries 数据集上我则直接使用 Apriori 算法进行分析。

3　实验及结果

3.1　模拟数据集

我将支持度和置信度的阈值分别设置为 0.1 及 0.6，蛮力算法的运行结果根据支持度和置信度排序后为：

[1]本次作业的主要代码实现可参见附录 A.2

```
————频繁项集————
('scissors', 'stapler') , 0.15
('plate', 'stapler') , 0.15
('knife', 'stapler') , 0.15
('scissors', 'knife') , 0.15
('stapler', 'glue') , 0.16
('ele_oven', 'TV') , 0.17
('milk',) , 0.18
('apple',) , 0.18
('bread',) , 0.19
('orange',) , 0.21
('beer',) , 0.22
('PC',) , 0.22
('phone',) , 0.22
('glue',) , 0.23
('plate',) , 0.23
('scissors',) , 0.25
('fridge',) , 0.25
('ele_oven',) , 0.26
('knife',) , 0.27
('TV',) , 0.29
('stapler',) , 0.30

————关联规则————
('scissors',) --> ('stapler',) , 0.60
('scissors',) --> ('knife',) , 0.60
('plate',) --> ('stapler',) , 0.65
('ele_oven',) --> ('TV',) , 0.65
('glue',) --> ('stapler',) , 0.70
```

得到以上的参考结果后，我使用 Apriori 算法和同样的参数对模拟交易数据集进行分析，所得结果见表 1 和表 2。

表 1: Apriori 算法在模拟交易数据集下发现的频繁项集（按支持度排序）

频繁项集	支持度	频繁项集	支持度
stapler, scissors	0.15	PC	0.22
knife, scissors	0.15	phone	0.22
stapler, knife	0.15	glue	0.23
plate, stapler	0.15	plate	0.23
stapler, glue	0.16	scissors	0.25
ele_oven, TV	0.17	fridge	0.25
milk	0.18	ele_oven	0.26
apple	0.18	knife	0.27
bread	0.19	TV	0.29
orange	0.21	stapler	0.30
beer	0.22		

表 2: Apriori 算法在模拟交易数据集下发现的关联规则（按置信度排序）

关联规则	置信度
scissors –> stapler	0.60
scissors –> knife	0.60
plate –> stapler	0.65
ele_oven –> TV	0.65
glue –> stapler	0.70

对比蛮力算法和 Apriori 算法排序后的结果，容易发现二者一致，可以证明我实现的 Apriori 算法的正确性。

3.2 Groceries 数据集

确认 Apriori 算法实现的正确性后，我将其应用到真实数据上。考虑到 Groceries 数据集商品种类相对于实例较少的特点，我选择了较小的支持度（0.01），置信度选择为 0.2。运行结果参见表 3 和表 4。

表 3: Apriori 算法在 Groceries 数据集下发现的频繁项集（按支持度排序）

频繁项集	支持度	频繁项集	支持度
stapler, scissors	0.15	PC	0.22
knife, scissors	0.15	phone	0.22
stapler, knife	0.15	glue	0.23
plate, stapler	0.15	plate	0.23
stapler, glue	0.16	scissors	0.25
ele_oven, TV	0.17	fridge	0.25
milk	0.18	ele_oven	0.26
apple	0.18	knife	0.27
bread	0.19	TV	0.29
orange	0.21	stapler	0.30
beer	0.22		

表 4: Apriori 算法在 Groceries 数据集下发现的关联规则（按置信度排序）

关联规则	置信度
scissors –> stapler	0.60
scissors –> knife	0.60
plate –> stapler	0.65
ele_oven –> TV	0.65
glue –> stapler	0.70

4 结论

A 附录

A.1 模拟交易数据集的详细信息

我将模拟交易数据集的交易记录按照交易商品数分为 4 类，即 2、3、4、5 件。不同的商品件数按照不同的比例随机混合三类商品，具体混合规则可参见表 5。

表 5: 模拟交易数据集生成交易记录的混合规则

商品数	混合规则（括号中数字代表各类商品在记录中所占数量）
2	(2); (1, 1)
3	(3); (2, 1)
4	(4); (3, 1); (2, 1, 1)
5	(5); (4, 1); (3, 2); (2, 2, 1)

A.2 Apriori 算法实现代码

```
from sklearn.cross_validation import StratifiedKFold
from sklearn.datasets import make_classification
from sklearn.cross_validation import train_test_split
from sklearn.metrics import classification_report

from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.grid_search import GridSearchCV

import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

##################### making dataset ######################

n = 500                   # number of instances
n_f = 30                  # number of features
n_c = 3                   # number of classes
inf_f = int(0.6 * n_f)    # 60% real features
red_f = int(0.1 * n_f)    # 10% redundant features
rep_f = int(0.1 * n_f)    # 10% repeated features
random_seed = 1           # random seed for the experiments

X, Y = make_classification(n_samples=n, n_classes=n_c, flip_y=0.03,
                     n_features=n_f, n_informative=inf_f, n_redundant=red_f,
                     n_repeated=rep_f, random_state=random_seed)
```

```
X_train, X_test, Y_train, Y_test = \
    train_test_split(X, Y, test_size=0.8, random_state=random_seed)

#################### making dataset #####################

# directly train the model with training set & testing set
def exp_plain_train():
    model = DecisionTreeClassifier(random_state=random_seed)
    model.fit(X_train, Y_train)

    pred = model.predict(X_test)
    print(classification_report(Y_test, pred))

    score = model.score(X_test, Y_test)
    print('plain train score in testing set: ', score)
    score = model.score(X_train, Y_train)
    print('plain train score in training set: ', score)

# test model with cross-validation
def exp_cv(folds=10):
    kfolds = StratifiedKFold(Y, n_folds=folds, random_state=random_seed)
    model = DecisionTreeClassifier(random_state=random_seed)
    scores_train = []
    scores_test = []
    for train, test in kfolds:
        model.fit(X[train], Y[train])
        # pred = model.predict(X[test])

        score = model.score(X[test], Y[test])
        scores_test.append(score)
        score = model.score(X[train], Y[train])
        scores_train.append(score)

    mean_test = np.array(scores_test).mean()
    mean_train = np.array(scores_train).mean()
    print('avg score with cv folds %d in testing set: '
          % folds, mean_test)
    print('avg score with cv folds %d in training set: '
          % folds, mean_train)

    return mean_test, mean_train

def plot_cv():
    mean_tests = []
    mean_trains = []
    folds_sum = 11
    for i in range(2, folds_sum):
```

```python
        m1, m2 = exp_cv(i)
        mean_tests.append(m1)
        mean_trains.append(m2)

    # start to plot
    x = np.arange(2, folds_sum)
    total_width, n = 0.8, 2
    width = total_width / n
    x = x - (total_width - width) / 2

    plt.bar(x, mean_tests, width=width,
            facecolor='#9999ff', edgecolor='white', label=u'测试集')
    plt.bar(x + width, mean_trains, width=width,
            facecolor='#ffa07a', edgecolor='white', label=u'训练集')
    for x, y1, y2 in zip(x, mean_tests, mean_trains):
        plt.text(x - 0.05, y1 + 0.01, '%.2f' % y1, ha='center', va='bottom')
        plt.text(x+width - 0.05, y2 + 0.01, '%.2f' % y2, ha='center', va='bottom')

    plt.xlabel(u'折数')
    plt.ylabel(u'平均 f1-score')
    plt.ylim((0, 1.3))
    plt.legend()
    plt.savefig('report/img/cv_bar')
    plt.show()

# grid search
def exp_grid_search(folds=10):
    model = DecisionTreeClassifier(random_state=random_seed)
    param_grid = {'criterion': ['gini', 'entropy'],
                  'max_features': ['sqrt', 'log2', None],
                  'max_depth': list(range(3, 15)),
                  'presort': [True, False],
                  'splitter': ['best', 'random']
                  }
    grid = GridSearchCV(model, param_grid, cv=folds, scoring='f1_weighted')
    grid.fit(X, Y)

    print(grid.best_params_)
    print(grid.best_score_)

    export_graphviz(grid.best_estimator_, filled=True, out_file='report/img/gs.dot')

# bagging alg
def bagging(cv=True):
```

```
    bagging = BaggingClassifier(
        DecisionTreeClassifier(random_state=random_seed),
        n_estimators=5,             # number of models
        random_state=random_seed,
        bootstrap=True,
        max_samples=1.0,             # Bootstrap sample size radio
        bootstrap_features=True,
        max_features=1.0,            # Bootstrap feature usage radio
    )
    if cv:           # using cross-validation
        scores_train = []
        scores_test = []
        kfolds = StratifiedKFold(Y, n_folds=10, random_state=random_seed)
        for train, test in kfolds:
            bagging.fit(X[train], Y[train])

            score = bagging.score(X[test], Y[test])
            scores_test.append(score)
            score = bagging.score(X[train], Y[train])
            scores_train.append(score)

        mean_test = np.array(scores_test).mean()
        mean_train = np.array(scores_train).mean()
        print('avg score with cv folds 10 in testing set: ', mean_test)
        print('avg score with cv folds 10 in training set: ', mean_train)
    else:            # without cross-validation
        bagging.fit(X_train, Y_train)
        pred = bagging.predict(X_test)
        print(classification_report(Y_test, pred))

    # check the features extracted by each model
    plt.figure(figsize=(7, 5))
    f_n = 30
    x = list(range(1, f_n + 1))
    for i, f in enumerate(bagging.estimators_features_):
        print('model %d' % (i + 1), f)
        plt.scatter(x, f, label=u'子模型 %d' % (i + 1))
    plt.xlabel(u'特征编号')
    plt.xticks(list(range(0, 41, 5)))
    plt.ylabel(u'特征数值')
    plt.legend(loc=1)
    plt.savefig('report/img/bagging_feature_%d' % len(bagging.estimators_features_))
    plt.show()

```

```python
def plot_bagging():
    # results
    x = list(range(10, 101, 10))
    y = [0.694, 0.720, 0.726, 0.730, 0.730, 0.738, 0.738, 0.738, 0.738, 0.744]
    y_b1 = [0.626, 0.672, 0.702, 0.716, 0.704, 0.727, 0.735, 0.732, 0.736, 0.738]
    y_b2 = [0.668, 0.711, 0.738, 0.734, 0.744, 0.746, 0.738, 0.730, 0.740, 0.738]

    # ploting code
    plt.figure(figsize=(6, 4))
    ax = plt.gca()
    ax.plot(x, y_b1, color='#90EE90', linewidth=1.7, label=u'70%特征')
    ax.plot(x, y_b2, color='#ffa07a', linewidth=1.7, label=u'90%特征')
    ax.plot(x, y, color='#9999ff', linewidth=1.7, label=u'100%特征')
    ax.scatter(x, y, s=13, c='#9999ff')
    ax.scatter(x, y_b1, s=13, c='#90EE90')
    ax.scatter(x, y_b2, s=13, c='#ffa07a')
    ax.grid(color='b', alpha=0.5, linestyle='dashed', linewidth=0.5)
    plt.xlim((5, 105))
    plt.xticks(x)
    plt.xlabel(u'子模型数量')
    plt.ylabel(u'平均 f1-score')
    plt.legend()
    plt.savefig('report/img/bagging_kline')
    plt.show()

# boosting alg
def boosting(cv=True):
    boosting = AdaBoostClassifier(
        DecisionTreeClassifier(max_depth=3, min_samples_leaf=2, random_state=random_seed),
        n_estimators=15,   # number of models
        algorithm='SAMME',   # Advanced-Boosting
        random_state=random_seed
    )
    if cv:        # using cross-validation
        scores_train = []
        scores_test = []
        kfolds = StratifiedKFold(Y, n_folds=10, random_state=random_seed)
        for train, test in kfolds:
            boosting.fit(X[train], Y[train])

            score = boosting.score(X[test], Y[test])
            scores_test.append(score)
            score = boosting.score(X[train], Y[train])
```

```python
            scores_train.append(score)

        mean_test = np.array(scores_test).mean()
        mean_train = np.array(scores_train).mean()
        print('avg score with cv folds 10 in testing set: ', mean_test)
        print('avg score with cv folds 10 in training set: ', mean_train)
    else:        # without cross-validation
        boosting.fit(X_train, Y_train)

        pred = boosting.predict(X_train)
        print(classification_report(Y_train, pred))

        pred = boosting.predict(X_test)
        print(classification_report(Y_test, pred))

    # plot the relation between weights and error
    plt.figure()
    plt.xlabel(u'子模型权重')
    plt.ylabel(u'错误率')
    plt.plot(boosting.estimator_weights_, boosting.estimator_errors_)
    plt.savefig('report/img/boosting-weight-error-%d' % len(boosting.estimator_weights_))
    plt.show()

def plot_boosting():
    # results
    x = list(range(10, 101, 10))
    y1 = [0.621, 0.649, 0.649, 0.666, 0.666, 0.672, 0.672, 0.688, 0.688, 0.710]
    y2 = [0.608, 0.570, 0.596, 0.616, 0.635, 0.633, 0.629, 0.650, 0.664, 0.668]
    y1_tr = [0.860, 0.958, 0.984, 0.995, 0.996, 0.999, 1.000, 1.000, 1.000, 1.000]
    y2_tr = [0.904, 0.936, 0.960, 0.975, 0.988, 0.993, 0.993, 0.995, 0.997, 0.998]

    # plot testing results
    plt.figure(figsize=(6, 4))
    ax = plt.gca()
    ax.plot(x, y1, color='#9999ff', linewidth=1.7, label='SAMME.R')
    ax.plot(x, y2, color='#90EE90', linewidth=1.7, label='SAMME')
    ax.scatter(x, y1, s=13, c='#9999ff')
    ax.scatter(x, y2, s=13, c='#90EE90')
    ax.grid(color='b', alpha=0.5, linestyle='dashed', linewidth=0.5)
    plt.xlim((5, 105))
    plt.xticks(x)
```

```python
    plt.xlabel(u'子模型数量')
    plt.ylabel(u'平均 f1-score')
    plt.legend()
    plt.savefig('report/img/boosting_kline_test')
    plt.show()

    # plot training results
    plt.figure(figsize=(6, 4))
    ax = plt.gca()
    ax.plot(x, y1_tr, color='#9999ff', linewidth=1.7, label='SAMME.R')
    ax.plot(x, y2_tr, color='#90EE90', linewidth=1.7, label='SAMME')
    ax.scatter(x, y1_tr, s=13, c='#9999ff')
    ax.scatter(x, y2_tr, s=13, c='#90EE90')
    ax.grid(color='b', alpha=0.5, linestyle='dashed', linewidth=0.5)
    plt.xlim((5, 105))
    plt.xticks(x)
    plt.xlabel(u'子模型数量')
    plt.ylabel(u'平均 f1-score')
    plt.legend()
    plt.savefig('report/img/boosting_kline_train')
    plt.show()

if __name__ == '__main__':
    # exp_plain_train()
    # exp_cv()
    # plot_cv()
    # exp_grid_search()
    boosting()
    # plot_boosting()
    # bagging()
    # plot_bagging()
```